

100 →

```

202  #define CQ_LENGTH 256 /* Power of 2 */
      struct cq {
        int head;
        int tail;
        void **cqqa[CQ_LENGTH];
      };

      static __inline int cq_enqueue(struct cq *cqp, void *p) {
        int i, nt;

        if (cqp->head ==
            (nt = ((i = cqp->tail) + 1) & (CQ_LENGTH - 1))) {
          return 0; /* Queue is full */
        } else {
          (*cqp->cqqa)[i] = p;
          cqp->tail = nt;
          return 1;
        }
      }

      static __inline void *cq_dequeue(struct cq *cqp) {
        int h;
        void *p;

        if ((h = cqp->head) == cqp->tail) {
          return 0; /* Queue is empty */
        } else {
          p = (*cqp->cqqa)[h];
          cqp->head = ((h + 1) & (CQ_LENGTH - 1));
          return p;
        }
      }

208  extern int mbuf_map(struct cq **mbuf_alloc,
210                    struct cq **mbuf_dealloc, int mbuf_prealloc);
212  extern int mbuf_unmap();
214  extern int mbuf_pull(int mbufs, int timeout);
216  extern int mbuf_push(int tap_descriptor, int mbufs);
      extern int interface_tap(char *ifname,
        struct cq **detour, struct cq **revert, int mode);
218  extern int interface_untap(int tap_descriptor);

```

FIG 2

USER LEVEL

KERNEL

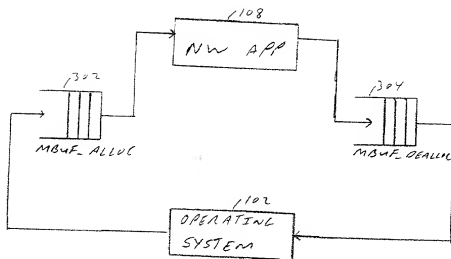


FIG. 3

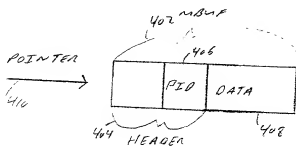


FIG. 4

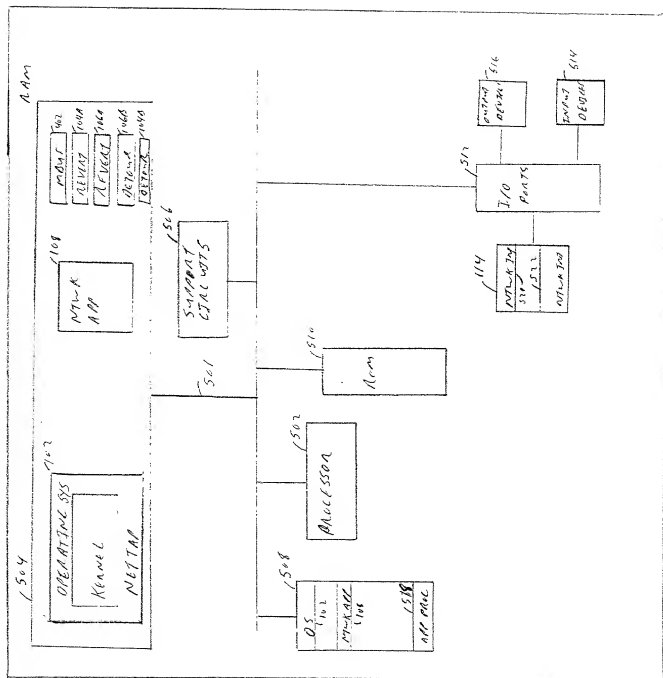


FIG. 5

500 →

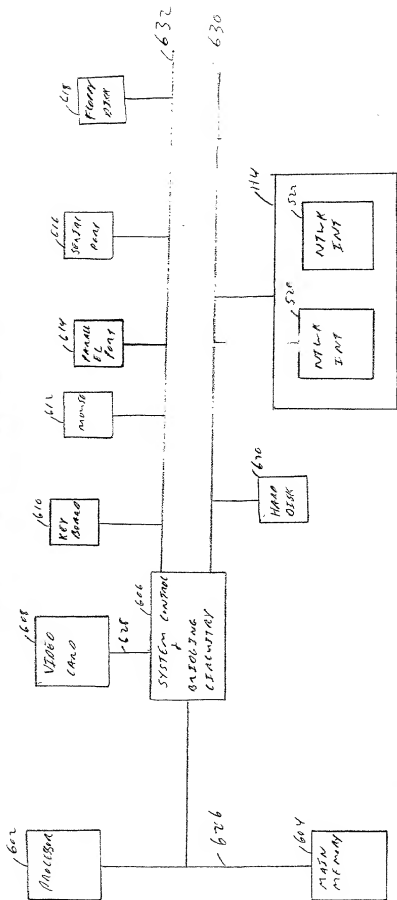


FIG. 6

```

#define ERROR(msg) {error(msg); exit(1);}
#define BUSY_WAIT_LIMIT 10

#define CLASS_DENY 0
#define CLASS_ACCEPT 1
#define CLASS_HOST -1

int network_application(struct mbuf *m) {
    ... process packet and return its classification ...
}

void drop(struct cq *dealloc, struct mbuf *m) {
    if (cq_enqueue(dealloc, m) == 0) {
        mbuf_push(-1, 0);
        if (cq_enqueue(dealloc, m) == 0) {
            ERROR("deallocating");
        }
    }
}

main () {
    int i, class, out, nempty, tin[2], tout[2];
    struct cq *alloc, *dealloc, *detour_tx[2],
        *revert_ip_in[2], *detour_ip_out[2], *revert_tx[2];
    char name[] = "tap0";
    struct mbuf *m;

    if (mbuf_map(alloc, dealloc, 0) != 0) {
        ERROR("Mapping mbufs");
    }

    for (i = 0; i < 2; i++) {
        name[i] = (i == 0) ? '0' : '1';
        if ((tin[i] = interface_tap(name, &detour_tx[i],
            &revert_ip_in[i], TAP_INPUT)) < 0) {
            ERROR("Tapping input");
        }
        if ((tout[i] = interface_tap(name, &detour_ip_out[i],
            &revert_tx[i], TAP_OUTPUT)) < 0) {
            ERROR("Tapping output");
        }
    }

    for (nempty = 0, i = 0; nempty++, i = (i + 1) & 1) {
        if ((m = cq_dequeue(detour_tx[i])) != 0) /* Rx pkt */
            nempty = 0;
        if ((class = network_application(m)) ==
            CLASS_ACCEPT) {
            /* Bridge pkt */
            out = cq_enqueue(revert_tx[(i + 1) & 1], m);
        } else if (class == CLASS_HOST) {
            /* Host input */
            out = cq_enqueue(revert_ip_in[i], m);
        } else {
            out = 0;
        }
        if (!out) {
            /* Drop if queue full or pkt denied */
            drop(dealloc, m);
        }
        if ((m = cq_dequeue(detour_ip_out[i])) != 0) {
            nempty = 0;
            /* Host output */
            if (cq_enqueue(revert_tx[i], m) == 0) {
                /* Tx pkt */
                drop(dealloc, m);
            }
        }
        if (nempty == BUSY_WAIT_LIMIT) {
            mbuf_pull(0, 0);
            nempty = 0;
            /* Wait for pkt */
        }
    }
}

```

700

FIG. 7